

Training Neural ODEs Using Fully Discretized Simultaneous Optimization*

Mariia Shapovalova * Calvin Tsay *

* Department of Computing, Imperial College London, London, UK
(e-mail: c.tsay@imperial.ac.uk).

Abstract: Neural Ordinary Differential Equations (Neural ODEs) represent continuous-time dynamics with neural networks, offering advancements for modeling and control tasks. However, training Neural ODEs requires solving differential equations at each epoch, leading to high computational costs. This work investigates simultaneous optimization methods as a faster training alternative. In particular, we employ a collocation-based, fully discretized formulation and use IPOPT—a solver for large-scale nonlinear optimization—to simultaneously optimize collocation coefficients and neural network parameters. Using the Van der Pol Oscillator as a case study, we demonstrate faster convergence compared to traditional training methods. Furthermore, we introduce a decomposition framework utilizing Alternating Direction Method of Multipliers (ADMM) to effectively coordinate sub-models among data batches. Our results show significant potential for (collocation-based) simultaneous Neural ODE training pipelines.

Keywords: Simultaneous dynamic optimization, nonlinear system identification, neural ODEs

1. INTRODUCTION

Data-driven dynamic models are increasingly prevalent in chemical and process systems engineering, providing useful alternatives to first-principles, physical models (Bhosekar and Ierapetritou, 2018; Thebelt et al., 2022). In particular, neural network-based models are popular surrogates in scheduling and control applications owing to their flexibility and representation power, e.g., as scale-bridging models (Tsay and Baldea, 2019). Neural networks can have various architectures and consist of learnable weights and biases that are optimized during training by minimizing a specified loss function. This is commonly achieved using gradient-based algorithms, which iteratively update the model parameters by taking steps in the negative direction of the gradient of the loss function (Sarker, 2021).

Neural Ordinary Differential Equations (Neural ODEs) (Chen et al., 2018) bridge neural networks with dynamical systems modeling, leveraging existing knowledge of ODEs. These models extend traditional networks to model unknown continuous-time dynamics by parameterizing the evolution of system states as a differential equation:

$$\frac{d\mathbf{x}(t)}{dt} = f_{\theta}(\mathbf{x}(t), t), \quad (1)$$

where $\mathbf{x}(t)$ represents the state at time t , and f_{θ} is the neural network parameterized by θ . Compared to more standard recurrent or convolutional neural networks, neural ODEs are flexible and can incorporate arbitrary time spacings. The framework has found numerous applications, including process control (Luo et al., 2023), reaction modeling (Sorourifar et al., 2023), and parameter estimation (Bradley and Boukouvala, 2021; Dua and Dua, 2012).

Despite their advantages, training Neural ODEs (or making predictions) requires solving an initial value problem (IVP) at each iteration using a numerical ODE solver. Given the initial condition $\mathbf{x}(t_0)$, the state at time $T \geq t_0$ is computed by solving the differential equation:

$$\begin{aligned} \mathbf{x}(T) &= \mathbf{x}(t_0) + \int_{t_0}^T f_{\theta}(\mathbf{x}(t), t) dt \\ &= \text{ODESolve}(\mathbf{x}(t_0), f_{\theta}, t_0, T), \end{aligned} \quad (2)$$

where ODESolve is a numerical IVP solver, and t_0 , T are the beginning and end of the integration interval, respectively. Training is typically based on the accuracy of the predictions $\mathbf{x}(T)$, requiring the numerical solution of (2) and backpropagation of gradients through the IVP solver at every iteration. These requirements lead to long training times of Neural ODEs (Lehtimäki et al., 2024).

Given the above, this work uses *spectral* numerical methods, specifically collocation, for the time integration of differential equations in Neural ODE training. Spectral methods offer several advantages: they are global as they approximate over the entire domain, display exponential convergence for smooth problems, and have better accuracy with a small number of points (Boyd, 2000). Spectral methods remain less explored than sequential methods in the context of Neural ODEs and have to date mostly been limited to approximating derivative targets (Roesch et al., 2021), or for non-simultaneous training (Quaglino et al., 2020). The novelty of this paper is that we show collocation can be employed in a *simultaneous* optimization approach, i.e., the system dynamics are solved as equality constraints rather than by iterative simulation, for fast and stable Neural ODE training. Furthermore, we show that the proposed method may produce more parsimonious models and is amenable to batching via ADMM.

* Support from a BASF/Royal Academy of Engineering Senior Research Fellowship is gratefully acknowledged.

2. NEURAL ODES FOR TIME SERIES

Neural ODEs can be applied in various contexts, e.g., in generative modeling or as implicit layers in larger models. We focus on the basic, control-relevant setting of modeling time-series data comprising observations $\{\mathbf{y}_i, t_i\}_{i=0}^{N-1}$, where $\mathbf{y}_i \in \mathbb{R}^d$ is the data vector at time t_i , and $T = t_{N-1}$ is the end of the time (and integration) interval. Our objective is to learn a parametric ODE model that, when integrated from an initial condition, results in a continuous trajectory $\mathbf{y}(t)$ approximating the observed data:

$$\mathbf{y}'(t) = f_{\theta}(\mathbf{y}(t), t), \quad \mathbf{y}(t_0) = \mathbf{y}_0, \quad t \geq t_0, \quad (3)$$

where $\mathbf{y}'(t)$ is the time derivative of the system state $\mathbf{y}(t)$ and f_{θ} is a neural network parameterized by θ . A trained Neural ODE model may also predict beyond the observed data interval $[t_0, T]$, provided the ODE solution remains valid and f_{θ} satisfies conditions such as Lipschitz continuity.

During training, the solution $\hat{\mathbf{Y}}$ is computed by numerically solving the IVP:

$$\hat{\mathbf{Y}} = \text{ODESolve}(f_{\theta}, \mathbf{y}_0, \mathbf{t}) \in \mathbb{R}^{N \times d}, \quad (4)$$

where $\mathbf{t} = \{t_i\}_{i=0}^{N-1}$ is the vector of time points matching the observations. The model parameters, θ , are learned by minimizing a loss function that captures the discrepancy between the predictions and observed data. The mean squared error (MSE) loss function is a common choice for continuous-output regression:

$$\mathcal{L}_{\theta}(\hat{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{N} \left\| \hat{\mathbf{Y}} - \mathbf{Y} \right\|_F^2, \quad (5)$$

where $\mathbf{Y} \in \mathbb{R}^{N \times d}$ is the matrix of observed values and $\|\cdot\|_F$ is the Frobenius norm. In summary, the goal of Neural ODE training can be formulated as computing θ such that (3) holds while minimizing (5).

2.1 Sequential ODE Solvers

In a typical Neural ODE training pipeline, we ensure (3) holds by solving the ODE system in every iteration. In this *sequential* approach, the solver computes $\hat{\mathbf{Y}}$ iteratively, e.g., by time stepping, until the end of the interval is reached at T . The simplest example is Euler's method, while more commonly used schemes are the Runge-Kutta methods, used as default solvers in `torchdiffeq` within PyTorch and `Diffraex` within JAX. We can generalize a step of a sequential numerical scheme as:

$$\mathbf{y}(t+h) = \mathbf{y}(t) + h \cdot \Phi(f, \mathbf{y}(t), t, h),$$

where h is the step size, f is the derivative function, and Φ denotes the (typically explicit) function that approximates the change in \mathbf{y} over the interval from t to $t+h$.

While this framework enables using tailored simulation methods for (3), using sequential ODE solvers for training poses several challenges. First, numerical errors can accumulate at each integration step, resulting in substantial global errors. Although adaptive solvers help control these errors, they add computational overhead and may still behave unpredictably on unseen data. Second, in addition to simulation CPU times, storing intermediate solutions for backpropagation requires significant memory. Despite the adjoint method (Chen et al., 2018) ensuring constant memory cost, it can substantially prolong training time.

3. SPECTRAL METHODS

As an alternative to the above, spectral numerical methods approximate the ODE solution as a linear combination of basis functions, e.g., trigonometric or orthogonal polynomials. The function coefficients are fitted over the integration domain, offering high accuracy and convergence rates for smooth problems (Boyd, 2000).

3.1 Collocation with Lagrange Interpolation

Collocation is a class of spectral methods in which an ODE is enforced at a set of discrete points, termed the *collocation grid*, which we introduce as $\xi = \{\xi_i\}_{i=0}^{N-1}$ in $[t_0, T]$. Under this framework, the approximate solution is

$$\tilde{\mathbf{y}}(t) = \sum_{i=0}^{N-1} \beta_i \phi_i(t),$$

where $\{\phi_i(t)\}_{i=0}^{N-1}$ represent the basis functions, $\{\beta_i\}_{i=0}^{N-1}$ are the coefficients to be determined, and $t \in [t_0, T]$.

We employ the barycentric form of Lagrange polynomials as basis functions due to its numerical stability and adaptability to diverse functions, including non-periodic behaviors (Berrut and Trefethen, 2004). The use of Lagrange polynomials also offers an implementation simplification due to the interpolation property, which ensures that the coefficients coincide with the true state values at the collocation grid, such that $\beta_i = \mathbf{y}_i$ at each collocation point. As a result, the approximated solution becomes:

$$\tilde{\mathbf{y}}(t) = \sum_{i=0}^{N-1} \mathbf{y}_i \ell_i(t), \quad (6)$$

where $\{\ell_i(t)\}_{i=0}^{N-1}$ are the Lagrange basis functions and $\{\mathbf{y}_i\}_{i=0}^{N-1}$ are the unknown coefficients, which are also the state values at each point ξ_i . We assume \mathbf{y} to be unknown, given the presence of noise in real-life systems. By treating all \mathbf{y}_i as coefficients (Berrut and Trefethen, 2004) and differentiating the interpolation formula (6), we obtain:

$$\tilde{\mathbf{y}}'(t) = \sum_{i=0}^{N-1} \mathbf{y}_i \ell'_i(t). \quad (7)$$

Substituting (6) and (7) into the Neural ODE (3) and evaluating at each collocation point ξ_i yields:

$$\sum_{j=0}^{N-1} \mathbf{y}_j \ell'_j(\xi_i) = f_{\theta} \left(\sum_{j=0}^{N-1} \mathbf{y}_j \ell_j(\xi_i), \xi_i \right), \quad i = 0, \dots, N-1. \quad (8)$$

This results in a system of nonlinear equations with respect to the unknowns $\{\mathbf{y}_j\}_{j=0}^{N-1}$. The system can be expressed in matrix form:

$$\mathbf{D}\mathbf{Y} = \mathbf{F}_{\theta}(\mathbf{Y}, \xi), \quad (9)$$

where:

- $\mathbf{D} \in \mathbb{R}^{N \times N}$ is the differentiation matrix with elements $D_{ij} = \ell'_j(\xi_i)$.
- $\mathbf{Y} = [\mathbf{y}_0, \dots, \mathbf{y}_{N-1}]^{\top} \in \mathbb{R}^{N \times d}$ is the matrix of unknown coefficients (true state values).
- $\mathbf{F}_{\theta}(\mathbf{Y}, \xi) = [f_{\theta}(\tilde{\mathbf{y}}(\xi_0), \xi_0), \dots, f_{\theta}(\tilde{\mathbf{y}}(\xi_{N-1}), \xi_{N-1})]^{\top} \in \mathbb{R}^{N \times d}$ contains the Neural ODE evaluated at each collocation point.

Using the barycentric formula (Berrut and Trefethen, 2004), the differentiation matrix is defined as:

$$D_{ij} = \begin{cases} \frac{w_j}{w_i} \frac{1}{\xi_i - \xi_j}, & \text{if } i \neq j, \\ - \sum_{\substack{k=0 \\ k \neq i}}^{N-1} D_{ik}, & \text{if } i = j, \end{cases}$$

where the weight is computed as:

$$w_i = \frac{1}{\prod_{\substack{k=0 \\ k \neq i}}^{N-1} (\xi_i - \xi_k)}.$$

The selection of the collocation grid significantly impacts the accuracy of the method. To mitigate errors caused by Runge's phenomenon, Chebyshev nodes of the second kind in $[-1, 1]$ are often used:

$$\xi_i = \cos\left(\frac{i\pi}{N-1}\right), \quad i = 0, \dots, N-1. \quad (10)$$

We refer the interested reader to Young (2019) for a comprehensive discussion of collocation grids.

4. PROPOSED METHODOLOGY

So far, we have converted the continuous ODE problem into a discrete algebraic system (9) by incorporating collocation and Lagrange interpolation. Our goal is to minimize the loss function (5) while enforcing that the collocation-estimated derivatives, computed as $\mathbf{D}\mathbf{Y}$, match the neural-network-predicted derivatives $\mathbf{F}_\theta(\mathbf{Y}, \boldsymbol{\xi})$ at the collocation points. The challenge arises because both the true values of the state values \mathbf{Y} and the parameters $\boldsymbol{\theta}$ of the neural network are unknown.

4.1 Simultaneous Approach

Our proposed approach is to incorporate the collocation system (5) as equality constraints in a single nonlinear optimization framework, where the objective function captures the discrepancy between the observed and optimized states, e.g., the MSE. By solving for \mathbf{Y} and $\boldsymbol{\theta}$ simultaneously, we effectively train the neural network from observed data while enforcing the collocation constraints.

The *simultaneous* approach in the context of collocation-based dynamic optimization can be further explored in the works by Tjoa and Biegler (1991); Kameswaran and Biegler (2006), where it is applied to the problem of parameter-estimation in differential equation systems.

4.2 Implementation

To implement this methodology, we utilize the Interior Point OPTimizer (IPOPT), which is well-suited for solving continuous, large-scale nonlinear optimization problems (Biegler and Zavala, 2009). For the software implementation, we call IPOPT through the open-source *Pyomo* algebraic modeling language. Recent research (Ceccon et al., 2022) demonstrates how neural networks can be represented as constraints within the Pyomo framework.

We initialize two optimization variable groups within Pyomo: *state variables* and *neural network parameters*.

- **State Variables**, \mathbf{Y}^* , represent the system's state at collocation time points. To expedite training, the state variables can also be initialized using smoothed observed data. These variables aim to approximate the true values \mathbf{Y} from the observed values \mathbf{Y}_{obs} .
- **Neural Network Parameters**, $\boldsymbol{\theta}$, include the weights and biases of the neural network.

The objective function captures the difference between the observed data and the state variables approximated by the collocation equations, instead of the output of ODESolve as in (4). We express the objective function as a combination of the MSE loss and regularization terms:

$$\mathcal{L}(\mathbf{Y}^*, \mathbf{Y}_{obs}) = \frac{1}{N} \|\mathbf{Y}^* - \mathbf{Y}_{obs}\|_F^2 + \lambda \|\boldsymbol{\theta}\|_2^2, \quad (11)$$

where

- $\mathbf{Y}^* \in \mathbb{R}^{N \times d}$ is the matrix of estimated state variables (the variables being optimized).
- $\mathbf{Y}_{obs} \in \mathbb{R}^{N \times d}$ is the matrix of observed values.
- $\boldsymbol{\theta}$ is the vector of neural network parameters.
- $\|\cdot\|_2$ and $\|\cdot\|_F$ are the Euclidean (L2) and Frobenius norms respectively.
- λ is the regularization parameter.

We enforce consistency between the neural network and the derivative of \mathbf{Y}^* at each collocation point ξ_i :

$$\sum_{j=0}^{N-1} \mathbf{y}_j^* l'_j(\xi_i) = f_\theta(\mathbf{y}_i^*, \xi_i).$$

Here, $l'_j(\xi_i)$ is the element D_{ij} of the differentiation matrix, so the left-hand side approximates the derivative of the optimized state values. The right-hand side is the output of the neural network.

4.3 Problem Formulation

We formulate the optimization problem as follows:

$$\min_{\mathbf{Y}^*, \boldsymbol{\theta}} \mathcal{L}(\mathbf{Y}^*, \mathbf{Y}_{obs}),$$

Subject to:

$$\begin{aligned} \text{Equality Constraints: } & \mathbf{D}\mathbf{Y}^* = \mathbf{F}_\theta(\mathbf{Y}^*, \boldsymbol{\xi}), \\ \text{Bounds: } & y_L^* \leq \tilde{\mathbf{y}}^* \leq y_U^*, \quad \theta_L \leq \boldsymbol{\theta} \leq \theta_U \end{aligned}$$

where:

- \mathcal{L} : is the loss function as described in (11).
- \mathcal{C} : $\mathbf{D}\mathbf{Y}^* = \mathbf{F}_\theta(\mathbf{Y}^*, \boldsymbol{\xi})$ is the matrix of equality constraints.
- \mathbf{Y}^* and $\boldsymbol{\theta}$ represent decision variables.
- y_L^*, y_U^* and θ_L, θ_U are the respective values for their lower and upper bounds

After the model is solved to optimality, the neural network can be used as the RHS of an ODE in a sequential or collocation-based solver in the post-training context.

4.4 Alternating Direction Method of Multipliers (ADMM)

One potential disadvantage of the above simultaneous framework is that the entire dataset must be handled in a

single optimization problem, while many training pipelines divide data into batches to alleviate computational or memory burden. We propose using the Alternating Direction Method of Multipliers (ADMM) to enable multi-batching by coordinating the training of separate submodels. For two ‘batches,’ the problem can be written as:

$$\begin{aligned} \min_{\theta_1, \theta_2} \quad & \mathcal{L}_1(\theta_1, \mathbf{Y}_1) + \mathcal{L}_2(\theta_2, \mathbf{Y}_2) \\ \text{s.t.} \quad & \theta_1 = \theta_2, \end{aligned}$$

where $\mathbf{Y}_1, \mathbf{Y}_2$ are the batches of data, θ_1, θ_2 are vectors containing parameters of each sub-model, and $\mathcal{L}_1, \mathcal{L}_2$ are the loss functions. Notice the ‘linking’ constraints $\theta_1 = \theta_2$ enforce a consensus model between the two data batches.

ADMM decomposes the above problem without the linking constraints by updating the optimization parameters and a dual variable (Lagrange multiplier) in an iterative manner (Boyd et al., 2010). Without the constraints, the problem is effectively decomposed into independent subproblems $\min_{\theta_i} \mathcal{L}_i(\theta_i, \mathbf{Y}_i)$. The loss functions for the subproblems are reformulated as follows:

$$\mathcal{L}_{\text{ADMM},i} = \mathcal{L}_i(\theta_i, \mathbf{X}_i) + \frac{\rho}{2} \left\| \theta_i - \bar{\theta}^{(k)} + \frac{\mathbf{u}_i}{\rho} \right\|_2^2, \quad \text{for } i = 1, 2,$$

where θ_i are parameters of submodel i , $\bar{\theta}^{(k)}$ are the consensus parameters at the k -th ADMM iteration, ρ is a scalar penalty strength, and \mathbf{u}_i are the dual variables associated with subproblem i . For two submodels, the consensus weights in the k -th iteration are given by:

$$\bar{\theta}^{(k)} = \frac{\theta_1^{(k)} + \theta_2^{(k)}}{2}$$

The dual variables for each submodel i are updated in each iteration using:

$$\mathbf{u}_i^{(k+1)} = \mathbf{u}_i^{(k)} + \rho(\theta_i^{(k)} - \bar{\theta}^{(k)}).$$

The model can be run for a fixed number of iterations or until convergence, which is defined by the magnitude of the primal residual, which we compute as:

$$r_{\text{primal}}^{(k)} = \sum_{i=1}^2 \|\theta_i^{(k)} - \bar{\theta}^{(k)}\|_2.$$

This ADMM framework allows us to train larger models by decomposing the problem into smaller subproblems, but can also improve the model generalization by learning from multiple trajectories simultaneously.

5. EXPERIMENTAL RESULTS

We compare the proposed collocation-based approach for training Neural ODEs to two benchmark sequential implementations: JAX (Diffrax) and PyTorch (torchdiffeq). Before training, we apply the following preprocessing steps:

- **Spacing:** We interpolate the training data to the chosen collocation grid, ensuring the data align with collocation points. The collocation grids are scaled according to the time range of the observed data.
- **Noise:** We simulate measurement noise by adding zero-centered Gaussian noise with $\sigma = 0.1$.
- **Initialization:** We utilize *Xavier* initialization for the neural network weights (Glorot and Bengio,

2010). For the simultaneous approach, the state variables \mathbf{Y}^* are initialized to the values of locally weighted polynomial regression (Cleveland and Develin, 1988).

5.1 Case Study: Van der Pol Oscillator

The forced Van der Pol Oscillator is a 2-D ODE system that can be represented as two coupled first-order equations:

$$\begin{cases} u' = v, & u_0 = 0 \\ v' = \mu(1-u^2)v - u + A \cos(\omega t), & v_0 = 1, \end{cases}$$

where u is the displacement, v is the velocity, ω is the angular frequency, μ is the damping parameter, and A is the external periodic force. For our experiments, we set the initial conditions as $u_0 = 0$ and $v_0 = 1$. The remaining parameters are chosen as $\mu = 1$, $A = 1$ and $\omega = 1$.

Training and Inference Procedure After training with our proposed collocation-based framework, the learned Neural ODE is used in a standard ODE solver (JAX Diffrax) for forward simulation. Figure 1 illustrates the prediction on both training and test ranges, showing that the collocation-trained Neural ODE captures the underlying dynamics effectively.

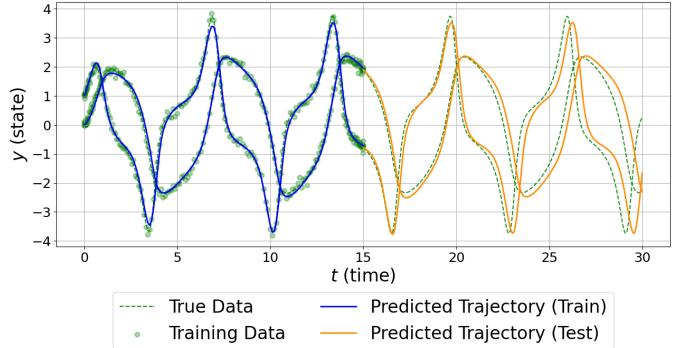


Fig. 1. Predictions of a model trained with collocation-based method (Pyomo) for the **Van der Pol Oscillator**. 200 training points and 200 testing points.

Pre-Training Strategies While we found that collocation-based training generally converged to good solutions, the *sequential* approach often resulted in premature local optima, perhaps owing to the feasible path approach. We therefore consider strategies for initial *pre-training* on a subset of the data (20%). As shown in Table 1, we use the notation *Model*[*Pre-training*] to denote different framework combinations, where *Model* is the strategy for the main training phase, and *Pre-training* is the strategy for the pre-training phase. The label *No* indicates that no pre-training was used. The standard method for pre-training uses the same model for both phases (JAX[JAX] or PyTorch[PyTorch]). We also explore a hybrid approach, where our Pyomo (collocation-based) model is used before continuing training with the benchmark models (JAX[Pyomo] or PyTorch[Pyomo]).

Table 1. Comparison of training frameworks. Smaller- and regular-size networks have layers of [2, 8, 2] and [2, 32, 2] nodes, respectively. Results averaged over 10 runs. Note the test MSEs are not available to the optimizers, which merely seek to minimize training MSE.

Model [Pre-training]	MSE Train	MSE Test	Time (s)
<i>Shorter Training Duration: Regular-Size Network</i>			
Pyomo [No]	0.0225	0.6139	7.144
JAX [JAX]	0.0553	1.0395	7.322
<i>Smaller-Size Network</i>			
Pyomo [No]	0.0374	0.9792	2.043
JAX [JAX]	0.1003	1.2908	8.705
<i>Longer Training Duration: Regular-Size Network</i>			
JAX [No]	2.7192	129.63	27.61
JAX [JAX]	0.0312	1.0954	23.85
JAX [Pyomo]	0.0098	0.4891	23.18
PyTorch [PyTorch]	0.4152	1.6728	25.14
PyTorch [Pyomo]	0.0111	0.4204	28.59

5.2 Performance Evaluation

Shorter Training Duration The collocation-based model exhibits fast convergence (approximately 7 seconds to train the regular-size network). We compare this with the accuracy achieved by the JAX model within the same training timeframe. As demonstrated in the *Shorter Training Duration* (top) section of Table 1, the Pyomo model achieves lower MSE during this time.

Smaller-Size Network We also observe that collocation-based training can produce models that achieve better performance with smaller network sizes, as shown in the *Smaller-Size Network* (middle) section of Table 1. Despite letting the JAX model train for longer using the smaller-size network configuration, it does not reach the performance level of the Pyomo model. This suggests that the improved optimization framework results in better training of the limited model parameters, leading to lower training and testing MSEs.

Longer Training Duration While training models with JAX and PyTorch for longer periods of time enhances their performance, we find they do not surpass the model trained with the collocation framework unless they are pre-trained with the latter, as detailed in *Longer Training Duration* (bottom) section of Table 1. Figures 2 and 3 show that the collocation-based training framework results in the fastest convergence in terms of both training and testing MSEs.

Hybrid Pre-Training We also demonstrate that the proposed framework can be used to pre-train alongside existing methods. For example, after the collocation-based training converges, we use its weights and biases to initialize a JAX model. Subsequently, the JAX model can continue training and further improve the results, as seen in Figure 4. The combination of the collocation-based training followed by training using JAX achieves the lowest training MSE scores, as seen in Table 1.

Batching Using ADMM Finally, we evaluate the performance of the proposed methodology for multi-batching

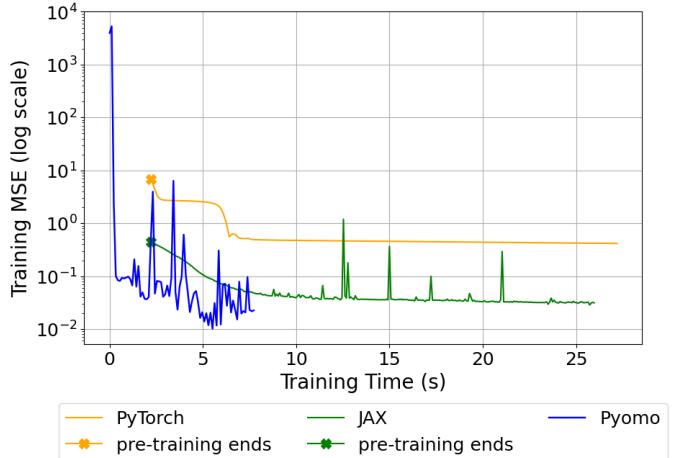


Fig. 2. **Training** MSE for three training frameworks. Note that the MSE values at intermediate training times are obtained by interrupting IPOPT’s runtime and may appear unstable.

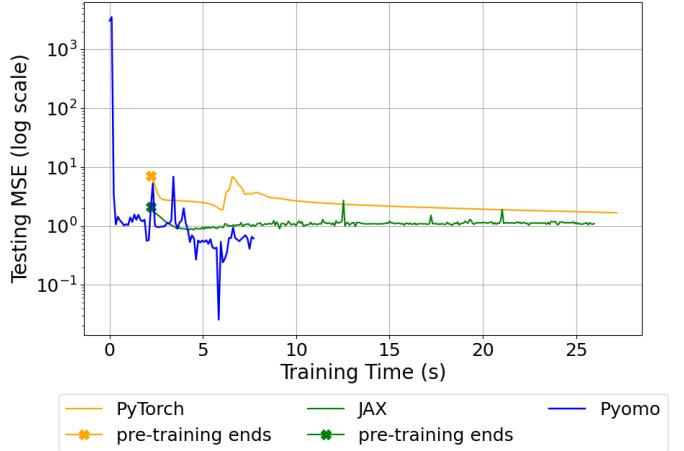


Fig. 3. **Testing** MSE for three training frameworks.

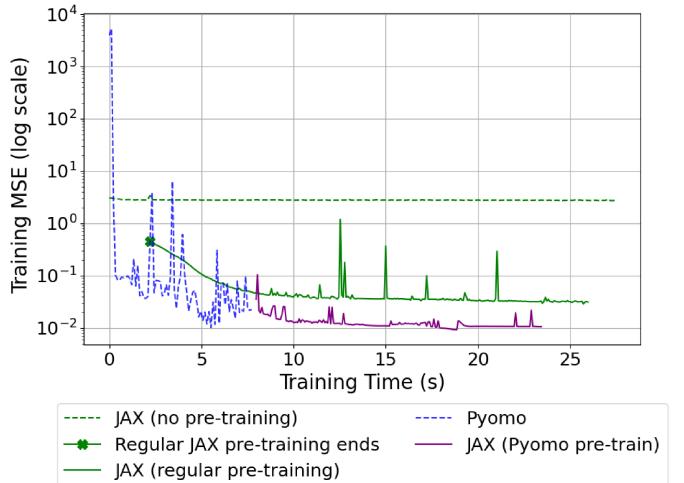


Fig. 4. **Training** MSE of JAX model with pre-training using ADMM. Figure 5 shows that ADMM is able to successfully coordinate the training of the two submodels, each containing half of the training data. The final MSE using the consensus weights of the ADMM-trained model

also surpasses the performance of a monolithic-trained model. This improvement is perhaps related to folklore observations that stochastic (batched) gradients may help escape local optima. Ultimately, the ADMM framework provides an avenue to train larger models more effectively.

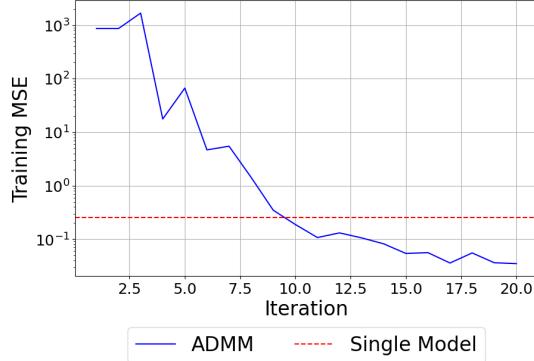


Fig. 5. ADMM (150 + 150 training points) vs Single Model (300 training points) Performance

6. CONCLUSIONS

In this work, we propose a collocation-based methodology using spectral methods for training Neural ODEs. By approximating the solution with Lagrange polynomials and enforcing differential equation constraints at collocation points, we recast the ODE problem as a system of algebraic constraints suitable for simultaneous optimization using IPOPT. Our experimental results on the Van der Pol Oscillator demonstrate that the proposed method achieves fast convergence and may enable more compact models compared to traditional sequential training approaches (e.g., implemented in JAX and PyTorch).

REFERENCES

- Berrut, J.P. and Trefethen, L.N. (2004). Barycentric Lagrange interpolation. *SIAM Review*, 46(3), 501–517.
- Bhosekar, A. and Ierapetritou, M. (2018). Advances in surrogate based modeling, feasibility analysis, and optimization: A review. *Computers & Chemical Engineering*, 108, 250–267.
- Biegler, L. and Zavala, V. (2009). Large-scale nonlinear programming using IPOPT: An integrating framework for enterprise-wide dynamic optimization. *Computers & Chemical Engineering*, 33(3), 575–582.
- Boyd, J.P. (2000). *Chebyshev and Fourier Spectral Methods*. Dover Publications, Mineola, NY, 2 edition.
- Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. (2010). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3, 1–122.
- Bradley, W. and Boukouvala, F. (2021). Two-stage approach to parameter estimation of differential equations using neural ODEs. *Industrial & Engineering Chemistry Research*, 60(45), 16330–16344.
- Ceccon, F., Jalving, J., Haddad, J., Thebelt, A., Tsay, C., Laird, C.D., and Misener, R. (2022). OMLT: Optimization & machine learning toolkit. *Journal of Machine Learning Research*, 23, 349.
- Chen, R.T.Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D.K. (2018). Neural ordinary differential equations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, volume 31, 6572–6583.
- Cleveland, W.S. and Devlin, S.J. (1988). Locally weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83(403), 596–610.
- Dua, V. and Dua, P. (2012). A simultaneous approach for parameter estimation of a system of ordinary differential equations, using artificial neural network approximation. *Industrial & Engineering Chemistry Research*, 51(4), 1809–1814.
- Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 9, 249–256.
- Kameswaran, S. and Biegler, L.T. (2006). Simultaneous dynamic optimization strategies: Recent advances and challenges. *Computers & Chemical Engineering*, 30(10), 1560–1575.
- Lehtimäki, M., Paunonen, L., and Linne, M.L. (2024). Accelerating neural ODEs using model order reduction. *IEEE Transactions on Neural Networks and Learning Systems*, 35(1), 519–531.
- Luo, J., Abdullah, F., and Christofides, P.D. (2023). Model predictive control of nonlinear processes using neural ordinary differential equation models. *Computers & Chemical Engineering*, 178, 108367.
- Quaglino, A., Gallieri, M., Masci, J., and Koutník, J. (2020). SNODE: Spectral discretization of neural ODEs for system identification. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*.
- Roesch, E., Rackauckas, C., and Stumpf, M.P.H. (2021). Collocation based training of neural ordinary differential equations. *Statistical Applications in Genetics and Molecular Biology*, 20(2), 37–49.
- Sarker, I.H. (2021). Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, 2(6), 420.
- Sorourifar, F., Peng, Y., Castillo, I., Bui, L., Venegas, J., and Paulson, J.A. (2023). Physics-enhanced neural ordinary differential equations: Application to industrial chemical reaction systems. *Industrial & Engineering Chemistry Research*, 62(38), 15563–15577.
- Thebelt, A., Wiebe, J., Kronqvist, J., Tsay, C., and Misener, R. (2022). Maximizing information from chemical engineering data sets: Applications to machine learning. *Chemical Engineering Science*, 252, 117469.
- Tjøa, I.B. and Biegler, L.T. (1991). Simultaneous solution and optimization strategies for parameter estimation of differential-algebraic equation systems. *Industrial & Engineering Chemistry Research*, 30(2), 376–385.
- Tsay, C. and Baldea, M. (2019). 110th anniversary: using data to bridge the time and length scales of process systems. *Industrial & Engineering Chemistry Research*, 58(36), 16696–16708.
- Young, L.C. (2019). Orthogonal collocation revisited. *Computer Methods in Applied Mechanics and Engineering*, 345, 1033–1076.